DPLR Package Users Guide

Introduction

The Data-Parallel Line Relaxation (DPLR) CFD code is an MPI-based parallel full three-dimensional Navier-Stokes CFD solver developed at NASA Ames Research Center. The code was written explicitly for the fast, robust and accurate solution of high-speed entry problems of interest to NASA. As such, generalized models for finite-rate reaction kinetics, thermal and chemical nonequilibrium, accurate high-temperature transport coefficients, and ionized flow physics are all incorporated into the code. DPLR also includes a large selection of generalized realistic surface boundary conditions, and "hooks" to enable efficient loose coupling with external thermal protection system (TPS) material response and shock layer radiation codes. The numerical methods employed, including the underlying data-parallel line relaxation algorithm, were chosen and developed to ensure optimal performance on distributed memory parallel machines. This makes the code widely portable to a variety of architectures, ranging from dedicated supercomputers to simple LINUX clusters and networked systems of desktop PC's or workstations. DPLR continues to evolve as a simulation tool, with new minor releases about every 2-3 months and major releases about every year. In addition, the code has been written in a way to encourage third-party modifications or improvements to the underlying package. Indeed, several of the features offered in v3.05.0 of the package were initially incorporated by third-party developers.

DPLR consists of four separate executables: DPLR2D (for axisymmetric and two-dimensional problems), DPLR3D (for three-dimensional problems), FCONVERT for grid pre-processing, and POSTFLOW for post-processing analysis. The following three chapters will detail the use of each of these programs. The current report is intended as a users manual only; an upcoming reference manual will discuss the theory behind the algorithms and physical models employed and a software programmers manual will discuss the implementation for those interested in modifying or improving the software package. The following chapters consist of users manuals for the CFD codes DPLR2D/DPLR3D, the pre-processor FCONVERT, and the post-processor POSTFLOW.

Required and Optional Software

There are only two required software packages that must be installed prior to DPLR:

(1) Fortran 90

DPLR is written entirely in Fortran 90, and as such requires a working f90 compiler on the destination machine.

(2) Message Passing Interface (MPI)

The code uses Message-Passing Interface (MPI) calls to facilitate interprocessor communications, and therefore an MPI library must be installed. Many machines have a native MPI package already installed, for the rest there is a free third party distribution called MPICH, which can be installed onto almost any machine architecture. For 64-bit architectures it may be preferred to use LamMPI, which is MPI-2 compatible.

In addition, there are several optional software packages that enhance the utility or performance of the package, but are not required for successful installation:

(1) FXDR

The *fxdr* libraries are freely available from:

http://meteora.ucsd.edu/~pierce/fxdr_home_page.html.

These libraries provide a Fortran-based interface to the native XDR (eXternal Data Reference) calls on all UNIX/LINUX machines. XDR enables the creation of platform independent binary files, which greatly enhances the portability of generated datasets (e.g. restart and grid files). The code can be compiled without the *fxdr* libraries, however in that case all restart and grid files must be written in either ASCII or machine-specific native binary format.

(2) TECIO.A

The Tecplot® I/O libraries are included with Amtec's Tecplot® visualization software, and may be available for free from the Amtec website:

http://www.tecplot.com/

There are two versions available, "`tecio.a`" for 32-bit architectures and "`tecio64.a`" for 64-bit architectures. These libraries are used by POSTFLOW to created Tecplot binary datafiles for post-processing output. The code can be compiled without these libraries if they are not available.

(3) LIBGOTO (BLAS routines)

DPLR makes use of several BLAS routines for matrix-vector and matrix-matrix manipulations. If such libraries are available on the target machine using them will lead to small (~20-25%) performance improvements in the overall runtime of the code. BLAS libraries are generally available from compiler makers as a part of their mathematical libraries for a nominal fee. In addition, several freeware sources exist. In particular, for Pentium or AMD architectures a freeware distribution called *libgoto* is available from:

http://www.tacc.utexas.edu/resources/software/

Installation

DPLR is typically distributed as a gzipped tar file "`dpcodeV3-05-0.tar.gz`". The first step is of course to unzip and untar the file. This leads to a directory structure given by:

`dpcodeV3-05-0/:`

```
cfdinput/       cfdlib/         config*         defs/
dplib/          dplr2d/         dplr3d/         fconvert/
include/        makefile        post/           utilities/
```

The contents of the directories and files is as follows:

cfdinput/
This directory contains physical modeling datafiles used by DPLR during execution. The contents of the directory will be discussed further below.

cfdlib/
This directory contains subroutines that are common to DPLR2D and DPLR3D.

config*
Configuration script used to setup the makefile for the specific machine architecture. See below for use.

defs/
This directory contains makefile templates for the various supported machines.

dplib/
This directory contains subroutines that are common to the entire package.

dplr2d/
This directory contains subroutines unique to the DPLR2D code.

dplr3d/
This directory contains subroutines unique to the DPLR3D code.

fconvert/
This directory contains subroutines unique to the FCONVERT code.

include/
This directory contains modules, common blocks and other include files that are incorporated into the various executables.

makefile
Makefile for the package.

post/
This directory contains subroutines unique to the POSTFLOW code.

utilities/
This directory contains utility codes and scripts that are distributed with the package.

A description of the contents of these directories is beyond the scope of the current DPLR Users Guide. The specifics of the indiviual source files are discussed in detail in the DPLR Programmers Manual.

Once the package has been untarred, the next step is to configure the makefile for the machine. The script *config* is provided to do this. *config* has been written to detect several different common machine architectures and generate a tailored makefile for them. When *config* is run it will generate a file "`makefile.comm`" which contains all of the machine specific information. Note that the script is relatively crude, and some tailoring of the resulting makefile may be required to ensure correct pathnames, etc.

Assuming that all of the pathnames are correct in "`makefile.comm`" the next step is simply to type "make", which should generate all of the executable files in the package. Finally, typing "make install" will install all of the executables and required input files in a bin directory, ready for use.

DPLR2D/DPLR3D

DPLR2D and DPLR3D are the main CFD codes provided in this distribution. In practice, the two CFD codes share most of the same physics and numerics subroutines and libraries, and are separated only by a difference in the "driver" routines. Separate executables are provided mainly for performance; it is much faster to run a two-dimensional or axisymmetric problem using DPLR2D than with DPLR3D. However, the two codes are for the most part interchangeable; both share a common input deck format and about 90% of their subroutines. In this manual we will refer to the code as "DPLR," but will indicate those (few) areas where the behavior of DPLR2D and DPLR3D are different.

Running DPLR

DPLR is run from the command line via MPI. The user first prepares the input deck, either by starting from a similar case or by following the rules discussed in the following section. In order to execute the run, type:

```
mpirun –np X (-machinefile machine.inp) $path/dplr2d < dplr.inp
```

at the command line, where "`X`" is the number of processors to use, "`machine.inp`" is the name of the machine file (required by some MPI implementations), and "`dplr.inp`" is the name of the input deck. When DPLR is executed, diagnostic output will be echoed to the screen in order to provide feedback on the action(s) being performed. Any warning messages will also be echoed to the screen. If a fatal error is detected during execution, a descriptive message will be echoed to the screen and execution will terminate. DPLR always runs as a parallel code, even if run on a single processor. See the users manual for FCONVERT for more information on how and why to run your problem on multiple processors, and how to set up your input grid file.

If a machinefile is required by your computer architecture, it consists simply of an ASCII listing of available machine (node) names, followed by a colon and the number of processes to start on each node. In such cases it is the users responsibility to ensure that the nodes listed in the machine file are available for use and free of other jobs prior to submission. An example machinefile for a dual-processor workstation cluster might look like this:

node001:2
node002:2
node003:2

With this file jobs of up to six processors can be submitted. If a larger job is submitted using this machinefile, more than two processes will be started on some or all of the nodes (they will be "reused") which will almost certainly lead to extremely slow performance or possibly even hangs or crashes of the software,

The sample input deck presented in the following section replicates that for one of the sample problems ("Neptune") provided with this distribution. When DPLR3D is executed on this sample problem, the output is:

```
*******************************************
 dplr3d
 NASA Ames Version 3.05.0
 Maintained by Mike Wright;  last modified: 04/02/06
*******************************************

# Running on   8 processors
# --> Allocating   1 nodes to block   1
# --> Allocating   7 nodes to block   2
# --> Maximum load imbalance =  13.73%
# --> Input grid file hardwired for   8 processors


# Summary of enabled CPP compiler directives:
# --> AMBIPOLAR =  0
# --> PARKTEXP = 0.50
# --> NOHTC


****WARNING: CPP macro AMBIPOLAR = 0
             disables ambipolar diffusion

# Neptune Mechanism: 5 species, 5 reactions (Liebowitz 1973 & 1976) Model
# --> Species List: H2 H H+ He e
# --> Reaction rates from: neptune5sp_leibowitz76.chem
# --> Reaction Status: 1 1 1 1 1
# --> Keq Fit Used   : 0 0 0 0 0
# --> Park 1990 fits for Keq (n=10^16)
# --> Assume molecules created/destroyed at mixture Tve


# Catalytic wall BC enabled
# --> Constant accomadation coeff;  gamma = 1.000
# --> Fully catalytic to ion recombination

# Radiative equilibrium BC enabled
# --> Constant wall emissivity;   epsilon = 0.85
# --> Maximum wall temperature = 3000.00 K

# Rotational Equilibrium - Fully Excited

# Vibrational Equilibrium - SHO

# Electronic Energy Neglected
```

```
# Laminar Navier-Stokes Simulation
# --> Gupta-Style Collision Integrals & Yos Mixing Rule
# --> Fickian Diffusion (Mass Fraction Gradients); Schmidt Number = 0.50

# Ideal Gas Equation of State

# 3-Dimensional Flow

# Implicit - Data Parallel Line Relaxation; kmax =  4
# --> Using Global Timestepping

# Estimate   187MB stack memory required per PE

# Reading grid file: t250-Over-8PE.pgrx
# --> Reading block   1: grid cell size   32X  16X  64
# --> Reading block   2: grid cell size   48X  64X  64
# --> Total number of grid cells =   229376
# --> Computing grid dummy cells

# Freestream Reynolds    Number = 8.024E+04 (1/m)
# Freestream Frozen Mach Number = 3.715E+01
# Freestream Equil. Mach Number = 3.715E+01


nit =      1 rmsres =  1.0000000000000E+00  cfl = 1.0E-05
:
:
```

Sample Input Deck

A sample input deck for DPLR is shown below. A brief description of each of the flags, along with allowable settings, is provided in the following section. Detailed discussions of some of the more complex options follow. Additional examples of DPLR input decks can be found in the sample problems that are distributed with the software package; it is recommended that the user run through each of the provided examples after reading this chapter and examine the output of DPLR for each case.

```
INPUT DECK FOR DPLR2D/DPLR3D CODE

gname,fname,bname,rname,dname
'mygridname'
'myrestartname'
'mybcname'
'myradname'
'PATH/cfdinput/air5sp_park85.chem'

nblk     igrid     irest      ibcf     iradf     nfree     iinit
   1,        11,       11,       11,        0,        1,       -1

ivis     ikt     ikv   ivmod    idmod    itmod    islip    iblow
   1,      1,     11,       3,       3,       1,       0,       0

icatmd   ireqmd    twall      epsr    gamcat     pback     vwall
     0,       0,   5.0d2,   0.85d0,    1.0d0,   1.01d5,    0.0d0

ichem     ikeq     ivib     irot    ieex     iel    irad    ipen
    1,       9,       1,       2,      0,      1,      0,      0

   itrmod    itrans     trloc     trext    itshk
        0,        0,    1.0d0,    0.1d0,       0

    istop     nplot     iplot      iaxi      ires
      700,      100,        1,        1,       -2

    igdum       kbl       kdg    istate     iresv
        0,        0,        0,        0,        1

   xscale       ils     Le/Sc   LeT/ScT      prtl     prtlT
    1.0d0,        2,   0.70d0,   0.50d0,   0.71d0,   0.90d0

   tfinal      xxxx       rvr    resmin
   9.0d99,   1.0d+0,    1.3d0,  1.0d-20

   ispace     dxmin   slength     nxtot
        0,   1.0d-2,    1.3d0,     1000
```

```
================================================================
 GRID ADJUSTMENT/ALIGNMENT/MORPHING
================================================================

  igalign    ngiter    nalign
        0,      500,         1

   imedge  imradial     ngeom     xxxxx
        1,         2,        2,       20

 fs_scale   ds_mult   gmargin
      0.9,       3.0,      0.0

      ds1    cellRe     ds1mx     ds2fr
   0.0d+0,       1.0,   1.0d-4,      0.3

================================================================
 BLOCK #1
================================================================

     ntx     nty     ntz     iconr      isim     ifree    initi
      30,     30,      1,        -1,         1,         1,        1

     iflx      iord      omgi      ilim     idiss      epsi
        2,         3,    2.0d0,         1,         1,       0.3

     jflx      jord      omgj      jlim     jdiss      epsj
        2,         3,    2.0d0,         1,         0,       0.3

     kflx      kord      omgk      klim     kdiss      epsk
        2,         3,    2.0d0,         1,         1,       0.3

iextst    kmax   ildir    ibcu   iblag     ilt    ibdir     cflm
    -1,       4,       0,       1,      -1,      -1,         1,  1.0d20

  Boundary condition type [ibc]:
    imin imax jmin jmax kmin kmax
     14    3   26    1   19   19

================================================================
 Freestream Specification #1
================================================================

     irm    density    M/Re/V     cx       cy       cz
       1, 6.8096d-4,  4.100d1, 1.0d0,  0.0d0,  0.0d0

     Tin       Trin       Tvin       Tein
 2.650d2,  2.650d2,   2.650d2,  2.650d2
```

```
   turbi     tkref
  1.0d-3,    0.0d0

      cs         (Species order: N2 O2 NO N O)
 0.767000d+0
 0.233000d+0
 0.000000d+0
 0.000000d+0
 0.000000d+0

================================================================
 List of CFL numbers or timesteps for ramping
================================================================

.001
.01
.1
.3
1.
3.   3
10.  4
25.
50.
100.
250.
500.
1000.
-1
```

Summary of Input Flags

Input flags will be described in the order in which they appear in the input deck. A full description of some of the more complex options will be deferred until later sections or the reference manual.

Some of the flags and/or options presented below are present for future expansion of the capabilities of DPLR and are not currently used. These will be indicated as they appear.

I/O Filenames

These are external input files used by DPLR at runtime. These files can be specified using relative or absolute pathnames. Depending on the format of the file a standard suffix will be assumed – see Appendix A for details.

gname

Name of the input grid file. This file is required and must already exist when the simulation is begun.

`fname`

Name of the input restart file. This filename is required. The restart file will be created if this is a new simulation, and must already exist if a restart is requested.

`bname`

Name of the input boundary condition file, if any. Boundary condition files are not required, but are used for several reasons as discussed in the section "Setting Boundary Conditions" below.

`rname`

Name of the input surface radiation, file if any. This file is optional, and is read only if volumetric radiation data are input (`irad` = 1). If the file is not required, use "none" as the filename.

`dname`

Name of the input chemistry file. This file is required. See the section titled "Input Physical Modeling Data" below for more information. Note that, unlike the previously defined input files, it is important that the full path name (as opposed to the relative pathname) to this file be given in the input deck.

Global Modeling Flags

These flags are for values that remain constant for all blocks of the simulation.

`nblk`

Number of master grid blocks in the simulation (note that `nblk` will be less than or equal to the number of processors on which the job is actually run).

`igrid`

Specify the format of the input grid file (`gname`). This can be any of the formats written by DPLR, summarized here:

1	Parallel archival file (native unformatted)
11	Parallel archival file (XDR format)
21	Parallel archival file (ASCII)

See Appendix A for a complete list and description of the various file formats supported by the DPLR software package.

`irest`

Specify the format of the restart file (`fname`). The choices are identical to those for `igrid`:

1	Parallel archival file (native unformatted)
11	Parallel archival file (XDR format)
21	Parallel archival file (ASCII)

`ibcf`

Specify the format of the input BC file (`bname`), if any. The choices are identical to those for `igrid`, plus the option of no file:

0	Do not read a BC file
1	Parallel archival file (native unformatted)
11	Parallel archival file (XDR format)
21	Parallel archival file (ASCII)

`iradf`

Specify the format of the input radiation file (`rname`), if any. The choices are identical to those for `ibcf:`

0	Do not read a radiation file
1	Parallel archival file (native unformatted)
11	Parallel archival file (XDR format)
21	Parallel archival file (ASCII)

`nfree`

Indicate the number of freestream specification records in the input file. See below for a description of these records.

`iinit`

Specify how to initialize the simulation. Possible options for `iinit` are:

0	Start all blocks by initializing to specified freestream values
1	Restart from saved file
2	Start with a stagnant interior at low pressure
3	Start with artificial boundary layer in place
10	Block-by-block initialization using `iconr` flag
11	Restart from saved file, reset `nit` and `etime`

More information on solution initialization, along with specific recommendations for specific problem types, is given below.

`ivis`

Specify the equation set to solve. Possible choices are:

0	Euler simulation (neglect Navier-Stokes terms)
1	Laminar full Navier-Stokes simulation
2	Turbulent full Navier-Stokes simulation
11	Laminar Navier-Stokes simulation (thin-layer)
12	Turbulent Navier-Stokes simulation (thin-layer)

DPLR is by default a full Navier-Stokes solver, but it can be run in Euler mode by setting `ivis` = 0. In this setting the viscous subroutines are never called, and thus run time per iteration is increased significantly. It is important to note that running DPLR as an Euler code while specifying viscous boundary conditions is very unstable. DPLR will print a warning message if this condition is detected.

Running DPLR in thin-layer mode (`ivis` = 11, 12) is provided mainly for historical continuity with other codes, and is not recommended in general. DPLR always computes the full Navier-Stokes fluxes if `ivis` > 0, but subsequently zeroes out the cross terms if a thin-layer option is specified. Therefore, there are no time or memory savings obtained by running in thin-layer mode.

For the turbulent cases (`ivis` = 2, 12) the turbulence model to be employed is selected using the `itmod` flag as discussed below.

`ikt`

Specify the model used to compute translational thermal conductivity. An appropriate setting for `ikt` is required for all viscous simulations (`ivis` ≠ 0). Possible choices are:

1	Use the model that is consistent with `ivmod`
2	Use constant Prantdl number expression

See the definition of `ivmod` below for a brief description of the models employed. A more thorough description will be given in the reference manual. `ikt` = 1 is the preferred choice for all practical applications; `ikt` = 2 is provided mainly as a way to compare results to other heritage codes.

`ikv`

Specify the model used to compute vibrational thermal conductivity. An appropriate setting for `ikv` is required for all viscous simulations (`ivis` ≠ 0) with vibrational nonequilibrium (`ivib` = 1, 3, 4). Possible choices are:

1	Standard expression with e_v gradients
2	Hard sphere approximation with e_v gradients
11	Standard expression with T_v gradients
12	Hard sphere approximation with T_v gradients

A thorough description of the "standard" and "hard sphere" expressions will be given in the reference manual. In general the hard sphere approximation is provided only for comparison to legacy codes and should not be used. The choice between e_v and T_v gradients is somewhat arbitrary, and scales the resulting vibrational thermal conductivity (κ_v) by the vibrational specific heat ($C_{v_{vib}}$):

$$q_v = \kappa \frac{\partial T_v}{\partial \eta} \approx \kappa \frac{\partial T_v}{\partial e_v} \frac{\partial e_v}{\partial \eta} = \kappa' \frac{\partial e_v}{\partial \eta}$$

$$\kappa_v' = \kappa / C_{v_{vib}}$$

For most simulations there is little difference between `ikv` = 1 and `ikv` = 11. However, for cases where the flow is nearly completely dissociated, using energy gradients becomes slightly unstable since there is little energy in this mode. For this reason, `ikv` = 11 is the preferred choice for all practical applications.

`ivmod`

Specify the baseline model used to compute mixture viscosity and thermal conductivity. An appropriate setting for `ivmod` is required for all viscous simulations (`ivis` ≠ 0). Note that the thermal conductivity model can be overridden with the `ikt` flag discussed above. Possible choices of `ivmod` are:

1	Blottner/Wilke model with an Eucken relation
2	Sutherlands Law and constant Prandtl number

3	Yos approximate mixing rules
4	RESERVED
11	Blottner/Armaly-Sutton with an Eucken relation
12	Keyes' Equation and constant Prandtl number

Again, a thorough description of the different models will be given in the reference manual. However, a few quick notes are discussed here. The Blottner/Wilke model (`ivmod` = 1) is widely used in the reacting flow community, although it has been shown to be inaccurate at elevated temperatures. It is provided primarily for comparison with legacy codes. The Armaly-Sutton model (`ivmod` = 11) is a clear improvement over Blottner/Wilke, but requires composition-dependent tailoring of the free parameters for maximum accuracy. Sutherland's Law (`ivmod` = 2) is available only for perfect gas flows, and is a reasonable estimate at low to moderate temperatures. Keyes' Equation (`ivmod` = 12) was developed for low temperature air flows and should only be used for perfect gases where the temperature is very low (<100 K). Finally, the Yos mixing rule (`ivmod` = 3) has been shown to be a reasonable and general approximation to the true Chapman-Enskog fluxes, and as such is the preferred model for all reacting gas simulations.

`idmod`

Specify the model used to compute species diffusion coefficients. An appropriate setting for `ivmod` is required for all multi-species viscous simulations (`ivis` ≠ 0). Possible choices of `idmod` are:

1	Constant Lewis/Schmidt number
2	Bifurcation model
3	Self Consistent Effective Binary Diffusion
5	RESERVED
11	Constant Lewis/Schmidt number, ignore ambipolar
12	Bifurcation model, fits include ambipolar
13	Self Consistent Effective Binary Diffusion, ignore ambipolar

Again, a thorough description of the different models will be given in the reference manual. However, a few quick notes are discussed here. The constant Lewis/Schmidt number model (`idmod` = 1) is the simplest choice, and assumes that all species have the same diffusion coefficient. The value of the Lewis or Schmidt number is given by the settings of the `ilt` and `Le/Sc` flags below. This model is included because it is widely used and simple to implement, but it is usually inaccurate. However, fair results can be obtained for a given problem class by choosing a Schmidt number to match benchmark results at similar conditions from a more accurate model. The Bifurcation model (`idmod` = 2, 12) was developed to model boundary layer diffusion of carbon-based ablators, and was employed by Olynick et al. in the design of the Stardust probe. The model

can produce accurate results, but requires as input least-squares fit coefficients for each species which are (ideally) obtained by more accurate simulations. These coefficients are imported to DPLR via the chemical model (`*.chem`) file employed.

The preferred model for all multi-species calculations is the Self-Consistent Effective Binary Diffusion (SCEBD) model of Ramshaw and Chang (`idmod` = 3, 13), which has been shown to give results in good agreement with exact solutions of the Stefan-Maxwell equations. This model requires as input collision integral data for each binary interaction in the mixture. These data are imported to DPLR via the "`gupta.tran`" physical model file in the **`cfdinput/`** directory. However, one problem with this model is that it tends to be somewhat unstable for separated flows, particularly while the recirculation region is being formed. Therefore, for separated flows it is recommended to start the solution with `idmod` = 1 and an appropriate Schmidt number, and then switch to `idmod` = 3 once the flow structures have stabilized.

The values `idmod` = 11-13 instruct DPLR to negelect ambipolar diffusion for ionized flows, and are provided mainly for comparison to other heritage codes. If the flowfield is non-ionized these are equivalent to their single-digit equivalents.

`itmod`

Specify the turbulence model to be employed. An appropriate setting for `itmod` is required for all turbulent simulations (`ivis` = 2, 12). Possible choices are:

0	Laminar Flow
1	Baldwin-Lomax Model
1000	Spalart-Allmaras Model (no compressibility corr.)
1001	Spalart-Allmaras Model (Catris&Aupoix comp.)
1002	Spalart-Allmaras Model (Secundov comp.)
2001	Menter SST Model (no compressibility correction)
2002	Menter SST Model (compressibility correction #1)
2003	Menter SST Model (compressibility correction #2)

Again, a thorough description of the different models will be given in the reference manual. However, a few quick notes are discussed here. The Baldwin-Lomax model (`itmod` = 1) is a zero-equation (algebraic) model commonly employed for many simulations at all Mach numbers and flow regimes. Comparison to ground test and flight data has shown that the Baldwin-Lomax model provides reasonable results for attached flows with a favorable pressure gradient on both blunt and slender bodies. However, agreement with data on separated flows and flows with adverse pressure gradients is much worse. For these flows the Menter SST two-equation model with compressibility corrections (`itmod` = 2003) is recommended.

Finally, all turbulent flow simulations require models for turbulent transport properties. The turbulent viscosity is computed via either an algebraic expression (`itmod` = 1) or from the expression for turbulent dissipation. Thermal conductivities are computed via a constant turbulent Prandtl number (`PrT`, defined below), while turbulent diffusion is computed via a constant turbulent Schmidt number (`ScT`, defined below).

`islip`

Specify the model to be used for slip-wall boundary conditions. Possible choices are:

0	Disable wall slip
1	Maxwellian slip model

Again, a thorough description of the different models will be given in the reference manual. Slip walls are only valid when `ivis` ≠ 0, and are generally used to relax the standard viscous no-slip wall boundary condition in order to simulate near non-continuum flows. Slip walls are not generally employed for simulations in the hypersonic or supersonic continuum, and so the usual setting for this flag is `islip` = 0. DPLR has currently implemented velocity and temperature slip models, but not species density (mole fraction) slip conditions. It should be noted that the slip wall model in DPLR has not been fully validated at this time, and should be used with caution.

`iblow`

Specify the model to be used for blowing-wall boundary conditions. Possible choices are:

0	Disable wall blowing
1	Specified wall blowing velocity
2	Specified unit mass flow rate

DPLR provides several ways to implement mass blowing through a solid wall. At this point these models only work for viscous walls (`ivis` ≠ 0). The `iblow` flag is used to specify relatively simple models as shown above, whereby the user can specify either a constant velocity or unit mass flow rate through every cell on the surface. The velocity or mass flow rate is specified with the `vwall` flag. In either case the resulting velocity is assumed to be normal to the surface, and the composition and temperature of the blown gas are taken from surface boundary conditions. Values of `vwall` > 0 result in a blowing wall, while `vwall` < 0 results in a sucking wall. This option is not often used in practice, and thus the

usual setting is `iblow` = 0. More complex models, including ablation, pyrolysis or transpiration gas blowing, are also implemented in DPLR using the material response boundary conditions as discussed below.

`icatmd`

Specify the model to be used for wall catalysis. Possible choices are:

0	Disable wall catalysis
1	Constant γ homogeneous model
2	Constant γ, fully catalytic to ions
3-98	Material specific surface kinetics
99	Input material map
100	Supercatalytic wall
101-198	Supercatalytic with specified freestream
200	Mitcheltree CO_2 model
201	Enhanced Mitcheltree CO_2 model
300	Generalized CO_2 catalysis (Bose/Wright)

Catalysis here refers to the wall surface facilitating chemical reactions that can deposit energy on the surface. DPLR offers many different models for surface catalytic reactions, which are summarized here. A more detailed description of the underlying physical models is given in the reference manual. If `icatmd` = 0 is specified the wall is assumed to be non-catalytic, which means that the gradient of all species mole fractions at the wall is zero.

The simplest catalytic model is the so-called supercatalytic wall (`icatmd` = 100), in which it is assumed that the chemical composition at the wall is identical to that in the freestream. This model is often employed for design studies because it provides maximum chemical enthalpy deposition on the surface and thus is a conservative estimate. However, the supercatalytic wall model assumes that all surface processes are infinitely fast and are not rate or diffusion limited, which is not typically a realistic assumption. DPLR also offers an "enhanced" supercatalytic wall BC (`icatmd` = 101-198) in which the wall composition is specified by a given freestream input block (see below for more information on specifying freestream conditions). This option is most often used for simulations of high enthalpy ground test facilities, where the actual freestream composition is frequently dissociated, while a true maximum enthalpy recovery boundary condition requires that the gas composition at the wall be fully recombined.

The other models offered include rate limiting effects. The simplest model is to assume that atoms that reach the wall recombine to form molecules with some constant efficiency, known as the accommodation coefficient (γ). This model is enabled by setting `icatmd` = 1 and fixing a value for γ with the `gamcat` flag. In an ionized flow there will also be recombination of the ions as the reach the

surface. This rate is also governed by `gamcat` if `icatmd` = 1. However, for an electrically neutral surface it is generally assumed that the wall is fully catalytic to ion recombination, regardless of the recombination efficiency of neutrals. This model is accessed by setting `icatmd` = 2. It is important to note that only homogeneous surface reactions are supported in this model, i.e. for air the two possible neutral surface reactions are

$$N + N \rightarrow N_2$$
$$O + O \rightarrow O_2$$

DPLR also supports material specific catalysis models, in which the catalytic efficiency is temperature and reaction dependent. These models are accessed by setting `icatmd` = 3-98. The rates for different materials are generally experimentally obtained, and are given in "`catalysis.surf`" in the **`cfdinput/`** directory.

Setting `icatmd` = 99 enables a surface catalytic map, which is read into DPLR via the input BC file, specified using the `bname` flag above. Use of this map allows the user to specify pointwise material properties and is discussed below.

At the current time DPLR has limited support for generalized (heterogeneous) catalysis models. The Mitcheltree model (`icatmd` = 200), developed for CO_2 atmospheres, models the recombination pathway $CO + O \rightarrow CO_2$ via a pair of diffusion limited steps. An enhanced version of the Mitcheltree model (`icatmd` = 201), add the additional pathway $C + O_2 \rightarrow CO_2$, which is potentially important for higher flow enthalpies. These models have been employed for Mars Pathfinder analysis, but there is no experimental evidence to support the existence of the required surface reactions. Finally, a generalized parametric CO_2 surface reaction model is available (`icatmd` = 300) which models the competing processes

$$CO + O \rightarrow CO_2$$
$$O + O \rightarrow O_2$$

via a pair of competing reactions with a user-specified perference factor. This model is generally employed in sensitivity analysis; it should not in general be used for design work.

`ireqmd`

Specify the model to be used for surface radiative equilibrium. Possible choices are:

0	Disable radiative equilibrium
1	Constant emissivity (ε) model
3-98	Material specific ε

99	Input material map
101-198	Material specific ε with a maximum temperature wall

A radiative equilibrium wall is a common design model that assumes that all energy incident to the surface is reradiated to space according to the expression:

$$q_w = \varepsilon \sigma T_w^4$$

where ε is the surface emissivity (which may be a function of temperature) and σ is the Stefan-Boltzmann constant. In this model the wall temperature is a derived property, based on the incident heat flux (as opposed to an isothermal wall where the temperature is a prescribed quantity). Although this model neglects other terms in the surface energy balance (such as conduction into the interior), it provides a quick and dirty first estimate of the flight heating rate, especially for non-ablating materials, where the thermal conductivity is kept intentionally low.

Possible choices include setting a constant value for the surface emissivity given by `epsr` (`ireqmd` = 1). The user can also specify material-specific properties (`ireqmd` = 3-98) or an input material map (`ireqmd` = 99), options discussed in more detail in the section on `icatmd` above. Finally, it is possible to place a maximum wall temperature limit criterion in addition to the models discussed above (`icatmd` = 101-198). This is included primarily to better model physical material temperature limits during initial design analysis. The maximum temperature is specified using the `twall` flag, and the code will automatically switch between an isothermal and radiative equilibrium wall on a pointwise basis if this option is employed.

`twall`

Specify the wall temperature to be used for isothermal (or temperature-capped radiative equilibrium) wall simulations.

`epsr`

Specify the constant value of emissivity (ε) to be used for radiative equilibrium wall simulations (`ireqmd` = 1).

`gamcat`

Specify the constant value of catalytic efficiency (γ) to be used for catalytic wall simulations (`icatmd` = 1, 2).

pback

Specify the back pressure. Currently not used in DPLR.

vwall

Specify the wall velocity. This flag allows the user to impose a constant blowing (`vwall` > 0) or sucking (`vwall` < 0) when used with the `iblow` flag. In addition, this flag can be used to specify a fixed wall velocity in a viscous flow simulation, but this option should be used with caution.

ichem

Specify the model employed for chemical reactions in the gas phase. Possible choices are:

0	Frozen chemistry
1	Finite-rate chemistry

A frozen chemistry model means that no chemical reactions occur in the flowfield, while the finite-rate model uses Arrhenius style reaction kinetics to model the chemistry. Chemical reaction rates are taken from the "`*.chem`" file specified with the `dname` flag discussed above. DPLR does not support equilibrium chemistry at this time.

ikeq

Specify the model employed for computing equilibrium constants. This is required when `ichem` = 1. Possible choices are:

-1	No reverse reactions
1	Park 1985 fits
2	Mitcheltree 1994 fits
3	Park 1990 fits ($n = 10^{16}/\text{cm}^3$)
4	Park 1990 fits ($n = 10^{19}/\text{cm}^3$)
9	Computed from NASA LeRC (1994) thermodynamic data

By default in DPLR, forward reaction rates are computed via Arrhenius expressions, while backward rates are computed from an equilibrium constant:

$$k_b = k_f / K_{eq}$$

where k_b is the backward rate, k_f is the forward rate, and K_{eq} is the equilibrium constant for the reaction. Several curve fit models are offered for computing the

equilibrium constants, including those from Park and one from Mitcheltree (`ikeq` = 1-4). Curve fit coefficients for these models are read from the file "`park.keq`" file in the **`cfdinput/`** directory. These models are provided mainly for heritage reasons, and should not in general be used, since they tend to behave poorly if the temperature in the flowfield extends outside of the fit range, leading to instabilities for some simulations. The preferred method of computing equilibrium constants is to use the minimized Gibb's free energy method (`ikeq` = 9), where species enthalpy and entropy are computed using curve fit expressions given by Gordon and McBride and the final equilibrium constant is then determined via the van Hoff't equation.

Finally, it is possible to "turn off" reverse reactions completely by setting `ikeq` = -1. This is provided as a debugging tool and should not be used for any real simulations.

`ivib`

Specify the model employed for computing the vibrational energy component of the gas. Possible values of `ivib` are:

0	Neglect vibrational energy
1	Vibrational nonequilibrium, single T_v
2	Vibrational equilibrium using statistical mechanics
3	Complete thermal equilibrium using NASA LeRC curve fits
4	Two temperature model using LeRC curve fits ($T_r = T_{el} = T$)
5	RESERVED
11	RESERVED

The details of the various models are discussed further in the reference manual. Briefly, the user can choose to either neglect vibrational energy (`ivib` = 0), model in equilibrium with the translational component of the gas (`ivib` = 2, 3), or model it in thermal nonequilibrium (`ivib` = 1, 4). Vibrational nonequilibrium is not very important for such flows as low altitude hypersonic flight or some Shuttle reentry trajectories. For these cases it is frequently sufficient to model the flow in vibrational equilibrium. When this is the case, the preferred option is `ivib` = 3, which uses the LeRC (1994) thermodynamic curve fit data to compute the total energy of a molecule as a function of temperature. This method implicitly includes non-ideal effects such as electronic excitation and anharmonic oscillators to the extent that these data are part of the LeRC database.

On the other hand, for most planetary and high velocity Earth entry flows, vibrational non-equilibrium has a first-order impact on both convective and radiative heat transfer and must be considered. For these flows `ivib` = 1 usually employed. In this model it is assumed that the vibrational mode of the gas is out of equilibrium with the translational modes, but all species vibrational modes are

governed by a single characteristic temperature. This is certainly an approximation of the true physics of the flow, but given that little quantitative data is available for the vibration-vibration coupling terms that would be required to increase the fidelity of the current model this is considered to be good enough for engineering purposes. The user can also model nonequilibrium with `ivib` = 4, which is essentially a two-temperature adaption of the complete thermal equilibrium model developed by Gnoffo; however at this time this method will not work for ionized flows.

Finally, it is important to note that if `ivib` = 3 (complete thermal equilibrium) is specified the input values of `irot` and `ieex`, and `iel` are ignored.

`irot`

Specify the model employed for computing the rotational energy component of the gas. Possible values of `irot` are:

1	Rotational nonequilibrium, single T_r
2	Rotational equilibrium using statistical mechanics
3	Complete thermal equilibrium using NASA LeRC curve fits
4	Two temperature model using LeRC curve fits ($T_r = T_{el} = T$)
5	RESERVED
11	RESERVED

The details of the various models are discussed further in the reference manual. Unlike vibration, the rotational mode of the gas is assumed to be fully excited, and thus cannot be neglected for polyatomic species. The basic choice before the user is to model the rotational mode in equilibrium with the translational mode (`irot` = 2-4), or in nonequilibrium governed by its own unique temperature (`irot` = 1). Note that the complete thermal equilibrium model (`irot` = 3) and two temperature model (`irot` = 4) are identical to those described for the `ivib` flag above. In practice it is rarely necessary to solve for a nonequilibrium rotational energy, since the nonequilibrium region is so small. This feature is provided mainly for detailed radiation studies of high altitude flows.

`ieex`

Specify the model employed for computing the electronic energy component of the gas. Possible values of `ieex` are:

0	Neglect electronic energy
1	Statistical mechanics ($T_e = T$)
2	RESERVED

3	Complete thermal equilibrium using NASA LeRC curve fits
4	Two temperature model using LeRC curve fits ($T_r = T_{el} = T$)
5	RESERVED

The details of the various models are discussed further in the reference manual. For `ieex` = 1 the contribution of the electronic energy to the total is computed using statistical mechanics based on characteristic temperatures and degeneracies in the "`chemprops.spec`" file from the **`cfdinput/`** directory, and is assumed to be equilibrium with the translational mode. The other possible models are discussed in the description of `ivib` above.

`iel`

Specify the model employed for computing the free electron energy component of the gas. This flag is only used when an ionized flow is modeled. Possible values of `iel` are:

1	Coupled electron and translational modes ($T_e = T$)
2	RESERVED
3	Complete thermal equilibrium using NASA LeRC curve fits
4	Two temperature model using LeRC curve fits ($T_r = T_{el} = T$)
5	RESERVED
11	RESERVED

The details of the various models are discussed further in the reference manual. `iel` = 3 and 4 are identical to that discussed for `ivib` above. `iel` = 1 assumes that the energy of the free electron gas is governed by the translational temperature. It is not currently possible to model the free electrons as coupled to the nonequilibrium vibrational temperature of the gas.

`irad`

Specify the model employed for shock layer radiation modeling. Possible values of `irad` are:

0	No radiation model
1	Read pointwise $\Delta \cdot Q_R$ from a file
2	Optically thin emission (CN Violet)
3	Optically thin emission (CN Red)
4	Optically thin emission (CN V+R)
102-104	Same as 2-4 with input surface heating

DPLR does not compute shock layer radiation directly, but several hooks are provided for coupling, either loosely to external radiation transport codes or

tightly for optically thin emission. Typically, for weakly radiating flowfields, shock layer radiation is either neglected or computed in an uncoupled manner. For these cases, set `irad` = 0.

If the radiation field is known to be optically thin, DPLR supports tight coupling by computing the $\Delta \cdot Q_R$ source term at each volume cell using curve fits generated by comparison to more exact computations. Currently DPLR supports this option for CN radiation only (`irad` = 2-4). In this case DPLR reads the curve fit coefficients from the file "`emission.rad`" from the **`cfdinput/`** directory. In this option it is assumed that energy converted to radiation is instantly lost from the control volume. A slight improvement on this model can be achieved by including the surface radiative heating effects in the radiative equilibrium surface energy balance. This is accomplished by using (`irad` = 102-104) and reading the pointwise surface radiative heating from the radiation file (`rname`). See Appendix XX for a description of the format of this file, and Ref. Wright2005 for a complete description of the effects of optically thin radiation coupling. Note that optically thin emission for other species can easily be added to the code if desired.

Finally, for most radiating shock layers the gas is absorbing as well as emitting. For this case the radiation transport becomes non-local and is beyond the current capabilities of DPLR to compute. However, it is possible to use a loosely coupled approach, iterating between DPLR and a radiation transport code such as NEQAIR, to determine the resulting coupled flowfield. This is done by running an initial solution neglecting radiation effects (`irad` = 0), and then solving for the $\Delta \cdot Q_R$ source term at each volume cell using a radiation transport code. The resulting source terms are then read into DPLR via the radiation file (`rname`), and the solution is restarted using `irad` = 1. The process is then repeated until a fully converged solution is obtained. This methodology was discussed in detail by Olynick et al. See Appendix XX for a description of the format of the required radiation file.

`ipen`

Flag is provided for future expansion and is not used by the code at this time.

`itrmod`

Specify the model employed for turbulence transition modeling. This flag is used whenever a turbulent flowfield (`ivis` = 2, 12) is specified. Possible values of `itrmod` are:

0	Neglect transition, flow is fully turbulent
1	TANH transition function
2	Dhawan and Narashima model

3	Sigmoid function
100	Input transition map

DPLR does currently have the capability to predict transition, but several models are in place to force transition at a user-specified location. If a fully turbulent flowfield is desired (as is usually the case), specify `itrmod` = 0. Three different models are provided to offer a smooth transition from laminar to turbulent flow at a user specified location (`itrmod` = 1-3). These are described in more detail in the reference manual, and make use of the flags `itrans`, `trloc`, and `trext` discussed below to define the location and extent of the transition region.

A more flexible approach is to use an input transition map (itrmod = 100). In this approach the user constructs a transition map consisting of a turbulence intensity value ranging from 0 (fully laminar), to 1 (fully turbulent) at each surface point. These data are read into DPLR via the BC file (bname). See Appendix XX for a description of the file format. This method allows the user to effectively simulate local turbulent regions in a laminar flow. However it is important to note again that all of the models available in DPLR force, but do not predict, turbulent transition.

`itrans`

Specify the ordinate of the transition onset location. This flag is used whenever a turbulent flowfield (`ivis` = 2, 12) with an input transition model (`itrmod` = 1-3) is specified. Possible values of `itrans` are:

±1	Transition at specified constant *x* value
±2	Transition at specified constant *y* value
±3	Transition at specified constant *z* value

Setting `itrans` to a positive value implies transition proceeds with increasing ordinate, while setting it to a negative value implies that transition proceeds with decreasing ordinate.

`trloc`

Specify the transition onset location. This flag is used whenever a turbulent flowfield (`ivis` = 2, 12) with an input transition model (`itrmod` = 1-3) is specified. `trloc` is a real dimensional number tied to the value of `itrans` above. As an example, if `itrans` = 1 and `trloc` = 2.5, the code will initiate transition at a value of x = 2.5 m, with turbulent flow for larger values of x and laminar flow for smaller values.

trext

Specify the extent of transition. This flag is used whenever a turbulent flowfield (`ivis` = 2, 12) with an input tanh transition model (`itrmod` = 1) is specified. `trext` is a real dimensional number equal to the width of the tanh function from .01-.99. The other transition models do not permit user modification of the transition length, so the `trext` flag is not used in these cases.

itshk

Flag is provided for future expansion and is not used by the code at this time.

istop

Specify the number of iterations to run before stopping. Note that `istop` is a relative, rather than absolute, value. For example, if a simulation in DPLR is restarted after 500 iterations are already complete and `istop` = 100, the code will run 100 additional iterations, reaching completion after 600 total iterations.

nplot

Specify the frequency of restart file writes. DPLR will save a restart file periodically every `nplot` iterations during the solution (as long as the flag `iplot` > 0). In general `nplot` should be set to a value large enough that DPLR does not spend a large percentage of the runtime writing restart files, but small enough that a lot of work is not lost in the case the job quits for some reason.

iplot

This flag is used to control the redundancy of restart file writes. Possible options for `iplot` are:

0	Do not write a restart file
1	Write a single restart file
n	Save *n* – 1 prior restarts
-99	Force restart file write

Obviously `iplot` = 0 should not be used except for debugging purposes, since the results of the simulation will not be saved in this case. Setting `iplot` to a positive integer larger than 1 causes the code to save *n* – 1 previous restart files in addition to the current one. This can be useful in case the code blows up, or to perform convergence testing. For this case DPLR will append the iteration number of the restart file to the filename in order to distinguish them. For

example, if the restart filename (`fname`) is specified as "`sample.pslx`", `iplot` = 3, and `nplot` = 200, then after 1000 iterations the following files will exist:

```
sample.pslx
sample.pslx-800
sample.pslx-600
```

Where the most recent restart file has no iteration suffix, and older files have the iteration number at which they were created. Older restart files are deleted by DPLR interactively.

Finally, the user can force a restart file write even if the file contains NaN's. This is typically used only rarely only for debugging purposes on parallel machine architectures for which NaN's are not fatal errors.

`iaxi`

This flag is used enable axisymmetry in the simulation. This flag is applicable only for DPLR2D runs. Possible values of `iaxi` are:

0	Non-axisymmetric (2D)
1	Axisymmetric about *x*-axis
2	Axisymmetric about *y*-axis

DPLR2D simulates axisymmetric flows by solving the Navier-Stokes equations in cylindrical rather than Cartesian coordinates. This allows for an axisymmetric simulation in about the same total solution time as a 2D result. The rotation axis of the problem is always assumed to be either the *x*- or *y*-axis, as shown above. Note that DPLR2D simulations are always in the *xy*-plane, so rotation about the *z*-axis is not permitted.

`ires`

This flag is used the type of residual and convergence data that are tracked and output to the screen and to the convergence file. Possible values of `ires` are:

0	Do not output a convergence file
1	Output nit, global residual, and Δt
2	Output nit, global residual, and CFL number
3	Output nit, global residual, and CPU time
4	Output nit, global residual, and flow time
11	Output nit, block residual, and Δt
12	Output nit, block residual, and CFL number

13	Output nit, block residual, and CPU time
14	Output nit, block residual, and flow time
22	Output nit, global residual, and min/max CFL
32	Output nit, block residual, and min/max CFL

When DPLR is run, convergence information is always output to both standard out and a convergence file (`*.con`). The `ires` flag gives the user control over what data to write. In each case the first output is the iteration number (`nit`). This is followed by one or more L2norm residuals. The variable(s) for which the L2Norm is computed is controlled by the `iresv` flag discussed below. Depending on the setting of `ires`, the user can examine either the global residual (summed over all computational blocks in the simulation), or the block-by-block residuals. Block-by-block output is useful to determine which blocks in the simulation are converging slowly (or not converging at all). Note that block output is based on the master blocks, not the parallel decomposed blocks. Also note that for simulations with many master blocks the output data can get voluminous. For this reason block residuals are only written to the convergence file; the data reported to standard out are always global residuals.

The final output variable is either the computed Δt for that iteration, the current CFL number, the elapsed CPU time, or the elapsed flow time. Note that elapsed flow time is only a useful output for time accurate simulations. DPLR always uses a global (rather than local) time step, but does support block-by-block CFL limiting using the `cflm` flag discussed below; therefore options are provided to output both the minimum and maximum CFL number employed.

Finally, if the `ires` flag is entered as a negative number the output residual(s) are normalized by the computed residual in the first iteration. This is generally the preferred output format, but a word of caution is required: some problem simulations can have zero residual of certain variables in the first iteration (i.e. viscous flow over a flat plate when tracking density residual). In this case the residual will be normalized by a very small number, leading to seemingly large normalized residuals at later iterations.

`igdum`

This flag controls the computation of grid dummy cell coordinates. Possible values of `igdum` are:

0	Only compute if necessary
1	Always recompute
-99	Only compute if necessary, output debugging files
99	Recompute and output debugging files

When a grid file is created by FCONVERT, dummy cell values are not computed, because FCONVERT does not have information about the correct boundary conditions to enforce at each grid face. When such a grid is read into DPLR for the first time the code will automatically generate the correct dummy cell coordinates for each block based on the supplied BC's, and send the correct data to all processors in the simulation. DPLR will then overwrite the stored grid file to include the dummy cell information. The preferred setting for this flag is `igdum` = 0, since there is in general no need to recompute dummy cells once they have been formed. However, it is essential to recompute the dummy cells (`igdum` = 1) if the block boundary conditions are changed. One common need for this is if a setup error is detected in the boundary conditions.

The other options (`igdum` = ±99) are provided mainly for developers to check for errors in the dummy cell computations. When these options are selected the code will generate a series of "`fort.xx`" files that consist of plot3d grid files including dummy cells for each parallel block in the simulation, where *xx* is equal to 50 + the processor number. For large simulations a lot of files will be generated. Again, these options are meant for code developers and should not in general be employed by users.

`kbl`

This is a rarely used flag that allows the user to zero out the body-normal added dissipation term in the boundary layer. If `kbl` is a positive integer the body-normal eigenvalue will be zeroed out for the `kbl` cells nearest to each solid wall in the simulation, and smoothly increased to the specified value. The flag is rarely used because it is seldom necessary or correct to have a non-zero value of the added dissipation term in the body-normal direction. See the reference manual for more details.

`kdg`

This flag is not meant for regular users and should be left equal to zero.

`istate`

This flag is for future expansion of the package and is not used at this time.

`iresv`

This flag controls the residual variable(s) that are tracked by DPLR. Possible choices of `iresv` are:

1	Total density
2	Velocity
3	Energy
4	Turbulence variables
-*n*	Conserved variable #*n*

The most common output is to sum the L2Norm of all species densities (`iresv` = 1), however DPLR also permits a sum over velocity components (`iresv` = 2), energy equations (`iresv` = 3), or turbulence variables for the Spalart-Allmaras or Menter SST model (`iresv` = 4). Finally it is possible to track the residual of a single equation by using a negative integer for `iresv`. For example, for a 5-species 2D simulation the residual in the *u* momentum component can be tracked with `iresv` = -6. Since DPLR is a fully coupled code (with the exception of some turbulence models), convergence of one variable is typically dependent on convergence of the others, which somewhat limits the utility of single variable residual. However it can be useful for unstable simulations, since the offending equation will generally "blow up" before the others.

`xscale`

This flag is used to scale the input grid at runtime. `xscale` should be used with great caution, since it is possible to end up multiply scaling the grid file if the user is not careful. The recommended setting is 1.0 (no scaling). DPLR will print a warning message if the value is not set to unity.

`ils`

Determines whether the input constants governing laminar (`Le/Sc`) and turbulent (`LeT/ScT`) diffusion coefficients are to be interpreted as Lewis or Schmidt numbers. Possible values for `ils` are:

1	Lewis Number
2	Schmidt Number

`Le/Sc`

Value of the laminar Lewis or Schmidt number to be employed in the simulation. The Lewis (*Le*) and Schmidt (*Sc*) numbers are defined as

$$Le = \rho D/\left(\kappa C_p\right); \qquad Sc = \mu/\left(\rho D\right)$$

This parameter is relevant for viscous simulations (`ivis` ≠ 0) with constant Schmidt number diffusion (`idmod` = 1, 11). Choosing a constant Schmidt number

is typically preferred. The appropriate value varies depending on the target destination and entry velocity, but is generally in the range of 0.4-0.7. Note that as stated above the preferred option is to model multispecies diffusion coefficients (`idmod` = 3), in which case this flag is not used during the simulation.

`LeT/ScT`

Value of the turbulent Lewis or Schmidt number to be employed in the simulation. Definitions are the same as for the laminar quantities if the laminar viscosity and thermal conductivity are replaced by their turbulent counterparts. This parameter is relevant for turbulent viscous simulations (`ivis` = 2, 12) regardless of the setting of `idmod`; there is no available model for variable Schmidt turbulent number available at this time. The computed catalytic heating in a turbulent reacting flow is strongly sensitive to the chosen value of the turbulent Schmidt number (decreasing Sc_T leads to increased heating); unfortunately there is no literature discussing the appropriate value to use for wall bounded flows. Values in the range of 0.5-1.0 are commonly employed; a value of 0.7 has been baselined for the Mars Science Laboratory.

`prtl`

Value of the laminar Prandtl number to be employed in the simulation. The Prandtl number (Pr) is defined as

$$\mathrm{Pr} = \rho\mu/\left(\kappa C_p\right)$$

This parameter is relevant for viscous simulations (`ivis` ≠ 0) with constant Prandtl number thermal conductivity model (`ivmod` = 2, 12 or `ikt` = 2). In general these models should only be selected for perfect gas (non reacting) low temperature flows; therefore the value of `prtl` is usually not relevant. For low temperature air flows a value of 0.72 is appropriate.

`prtlT`

Value of the turbulent Prandtl number to be employed in the simulation. The turbulent Prandtl number ($\mathrm{Pr_T}$) is defined as

$$\mathrm{Pr}_T = \rho\mu_T/\left(\kappa_T C_p\right)$$

This parameter is relevant for all turbulent viscous simulations (`ivis` = 2, 12) regardless of the turbulence or laminar conductivity model. A value of `prtlT` = 0.9 is usually selected (albeit without much justification).

`tfinal`

For time accurate simulations this is the final flow time desired, in seconds. This flag is not used for steady-state problems.

`xxxx`

Not used in DPLR at this time.

`rvr`

Viscous overrelaxation parameter. In general this should not be changed from its default value of 1.3.

`resmin`

The solution will be time-marched until the L2Norm residual reaches this level. For normalized residuals a value of 1×10^{-8} or lower will usually ensure a fully converged solution. If you prefer that the solution always runs to a specified number of iterations (`istop`), set `resmin` equal to some very small value (i.e. 1×10^{-20}).

`ispace`

This and the following three flags are only used to perform 1D space marching simulations (such as simulations of shock tube flows). Possible values of `ispace` are:

0	Disable space marching
1	Enable space marching

The space marching routine is seldom used in practice, but it provides an extremely fast and efficient tool to perform simulations of 1D flows (such as shock tubes) with complex models. The marching algorithm employs an explicit 4^{th}-order Runge Kutta integrator. If this option is selected many of the other options of the code are not used.

`dxmin`

Set the minimum *x*-spacing for the space marching routine. This option is only used when `ispace` = 1.

`slength`

Set the total marching distance for the space marching routine. This option is only used when `ispace` = 1.

`nxtot`

Set the total number of cells for the space marching routine. This option is only used when `ispace` = 1.

Grid Adaption Flags

The following flags are used to set up and initialize the optional automatic grid alignment feature in DPLR. See below for more information about grid alignment.

`igalign`

This flag is used to enable and set the type of grid adaption requested. Possible choices are:

0	Do not perform grid alignment
1	Perform basic grid alignment
2	Recluster grid only; no alignment
3	Smooth outer boundary only; no alignment
11	RESERVED

Basic grid alignment (`igalign` = 1) instructs DPLR to attempt to move the outer boundary of the grid to be in better alignment with the bow shock wave and redistribute the volume points accordingly. More information about this option is given in the section on grid alignment below, but it is important to note that the user should first converge an initial solution such that the shock has reached its final location before attempting an alignment. Because of this, it is an error to set `igalign` to 1 other than during a restart from an existing solution.

The basic grid alignment procedure consists of three main steps: moving the outer boundary to just beyond the shock location, smoothing the outer boundary surface, and redistributing the interior grid points. The second and third steps can be run independently if desired by setting `igalign` = 2 or 3, respectively. Since these settings do not involve locating the shock, they can be performed at any time during the solution (even in the first iteration of a new problem).

. **`ngiter`**

This flag controls the frequency at which a grid alignment is performed. The first alignment always occurs on a restart prior to running the first iteration, and subsequent alignments (with the total number given by `nalign`) are performed every `ngiter` iterations.

`nalign`

Set the total number of adaptions to perform during the simulation. Generally speaking 2-3 alignments are sufficient to produce high quality solutions. Setting `nalign` to 0 is the same as setting `igalign` to 0.

`imedge`

This flag controls the method used to locate the bow shock in the simulation. Possible choices are:

1 Align to a constant Mach number contour

At the current time only Mach number based adaption is supported, because it proved to be the most robust and generally useful choice in preliminary testing. Additional shock detection methods could be added if desired.

`imradial`

This flag controls the type of wall spacing to employ durnig the reclustering of interior points. Possible choices are:

1 Constant cell Reynolds number wall spacing
2 Use a constant wall spacing

If cell Reynolds number spacing is specified (`imradial` = 1) the code will attempt to select a wall spacing at every surface point that produces a cell Reynolds number specified by the `cellRe` flag. The cell Reynolds number is defined as:

$$\mathrm{Re}_c = \left(\frac{\rho c}{\mu}\right)_w \Delta\eta$$

This will of course result in a wall spacing that varies over the body surface. The flags `ds1` and `ds1mx` defined below can be used to limit the minimum and maximum allowable wall spacing, respectively.

If a constant wall spacing is specified (`imradial` = 2) the code will set the wall spacing at all surface locations to be equal to the value specified by the flag `ds1`. If `ds1` is set to zero, the current wall spacing will be used.

`ngeom`

This flag specifies the number of geometrically spaced points to place near the body surface during reclustering. If `ngeom` ≤ 2 a pure two-sided Vinokur stretching routine will be used. `ngeom` = 2 is the recommended setting for most problems.

`xxxxx`

This flag is not currently used in DPLR and is provided for future expansion.

`fs_scale`

This flag specifies the fraction of the freestream Mach number to pick as the adaption contour. This should be a value close to, but less than one. Smaller values of `fs_scale` lead to smoother grids, but increase the chance that the final outer boundary will not contain the entire shock. The recommended setting for most problems is a value in the range 0.90 ≤ `fs_scale` ≤ 0.95.

`ds_mult`

This flag specifies how much to grow the outer boundary beyond the location of the adaption contour. `ds_mult` is defined as a multiplier on the final grid spacing at the outer boundary determined via the reclustering routine. The recommended value for `ds_mult` is somewhat problem dependent, but values in the range of 2.5 – 3.0 seem acceptable for most problems.

`gmargin`

This flag specifies an optional additional padding or margin to add to the outer boundary of the grid. It is again defined as a multiplier on the final grid spacing at the outer boundary. For most problems `gmargin` can be set to zero. Note that the gmargin flag can cause the final outer boundary to be larger than the initial in certain cases. This is a feature in that it allows the boundary to grow if the shock is too close because of early adaption or insufficient `ds_mult`. However, the option should be used with some caution as any growth to the outer boundary is accomplished via simple linear extrapolation of the grid lines. This can lead to

large skewness and eventually grid folding if any of the body-normal lines are convergent.

ds1

This flag has different meanings depending on the setting of `imradial`. For `imradial` = 1 (cell Reynolds number spacing), `ds1` sets the minimum allowable wall spacing anywhere in the volume. For `imradial` = 2 (constant spacing), `ds1` sets wall spacing everywhere in the volume. For this option setting `ds1` = 0 causes DPLR to maintain the wall spacing in the current grid.

cellRe

This flag specifies the value of cell Reynolds number when `imradial` = 1. It is silently ignored for other values of `imradial`.

dsmx

This flag sets the maximum wall spacing allowed when cell Reynolds number spacing (`imradial` = 1) is employed. It is silently ignored for other values of `imradial`.

ds2fr

This flag sets the spacing at the outer boundary of the grid. `ds2fr` is expressed as a fraction of the spacing that would be used if an unconstrained (one-sided) Vinokur stretching algorithm were employed. Values of `ds2fr` around 0.35 work well for most simulations.

xxxxx

This flag is not currently used in DPLR and is provided for future expansion.

Block-Specific Flags

The following flags can be set differently for each block in the simulation. This includes things like explicit and implicit flux discretizations, inialization, and boundary conditions.

ntx, nty, ntz

Set the total number of computational cells in the *ijk* directions for this block. These should be set to the number of interior cells (not including dummy or ghost cells). Note that `ntz` is only used for 3D flows.

`iconr`

This flag has the same permissible values as `iinit` described above, and is used only for block-by-block initialization when `iinit` = 10. For all other cases the value of `iconr` is ignored.

`isim`

This flag allows the user to selectively eliminate master blocks from the simulation. Possible choices for `isim` are:

0	Do not include block in the simulation
1	Include block in the simulation

Note that the method used in DPLR to exclude blocks is currently very primitive; the code still allocates processors to those blocks as if they were being simulated, but the conserved variables are never updated. Because of this, exluding blocks in the simulation does not save on the computational intensity of the simulation, and it seldom used in practice. One of the few purposes of this option is to freeze problem blocks while the remainder of the solution is allowed to converge normally.

`ifree`

Specify the number of the freestream specification to use for this block. The total number of freestream specification records in the input deck is specified with the `nfree` flag discussed above. The format of each freestream record is given below. All freestream boundary conditions (`ibc` = 1) in this block will be initialized to the specified freestream conditions each time DPLR is run.

`initi`

Specify the number of the freestream specification to use to initialize this block. The total number of freestream specification records in the input deck is specified with the `nfree` flag discussed above. The format of each freestream record is given below. All volume cells in the interior of the block are initialized to the specified conditions when the problem is first run (`iinit` = 0), or when individual blocks are re-initialized (`iconr` = 0; `iinit` = 10).

`(ijk)flx`

This flag specifies the method to use to extrapolate the Euler fluxes in the *i*, *j*, or *k* directions. The method for flux extrapolation can be set separately in each computational direction. Possible choices are:

0	No flux evaluation
1	Upwind modified Steger-Warming with Δp
2	MUSCL Steger-Warming with Δp [p, c_s, $\vec{u}$, $\vec{T}$]
3	MUSCL Steger-Warming with Δp [ρ_s, $\vec{u}$, $\vec{T}$]
4	MUSCL Steger-Warming with Δp [p, c_s, $\vec{u}$, $\vec{e}_i$, T]
5	Pure 2nd order central difference
11	Upwind modified Steger-Warming without Δp
12	MUSCL Steger-Warming without Δp [p, c_s, $\vec{u}$, $\vec{T}$]
13	MUSCL Steger-Warming without Δp [ρ_s, $\vec{u}$, $\vec{T}$]
14	MUSCL Steger-Warming without Δp [p, c_s, $\vec{u}$, $\vec{e}_i$, T]

The primary method for flux extrapolation within DPLR is based on a modified form of the Steger-Warming algorithm, developed by MacCormack and Candler. This approach is considerably less dissipative than the original form, and has been shown to be extremely robust for a wide variety of hypersonic and supersonic flow simulations.

The flux evaulation can be "turned off" in any direction by setting `ijkflux` = 0, which can be useful to mimic the simulation of a lower-dimensional problem.

The setting `ijkflux` = 1, 11 are provided mainly for historical reasons, and uses a simple upwind stencil to compute the fluxes. The order of accuracy can be either first or second (set with the `ijkord` flag discussed below). In the case of second-order accuracy, a simple pressure gradient based switch is used to drop the accuracy to first order in the neighborhood of strong shock waves.

The settings `ijkflux` = 2-4 and 12-14 use a MUSCL-based adaptive stencil to attain higher-order accuracy via a more sophisticated approach. The difference between the selections is in the set of variables that are extrapolated to attain high-order accuracy, and whether a pressure gradient based switch is employed to smoothly transition from high order to first-order in the vicinity of strong shock waves. For most problems `ijkflux` = 2 is the recommended choice. The others are provided mainly for testing purposes. Note that extrapolation of the conserved variables via MUSCL limiting is extremely non-robust and is not offered as an option in DPLR. The various sets of primitive variables provided as options were found to be the most stable for problems of interest.

Finally, `ijkflux` = 5 invokes a simple central difference scheme (when used with `ijkord` = 2). This should not be used for problems which contain shock waves. No explicit dissipation scheme, such as those commonly employed in subsonic flow simulations, is provided in DPLR, so this method may not be very stable even for subsonic flows.

`(ijk)ord`

This flag specifies the nominal order of accuracy of the Euler flux extrapolation. Possible choices are:

1	First-order upwind
2	Second-order upwind biased
3	Third-order upwind biased

Generally the third-order upwind biased method (`ijkord` = 3) is recommended for all directions and all simulations in DPLR.

`omg(ijk)`

This flag specifies the value of ω (as defined by Yee) to employ in the MUSCL extrapolation scheme. The range of possible ω is dependent on the chosen order of accuracy of the schem (`ijkord`) and is given by the expression

$$1.0 \le \omega \le (3-\kappa)/(1-\kappa)$$

where κ = -1 for a second-order scheme and κ = 1/3 for a third-order scheme. The maximum value of κ is therefore 2 for a second-order scheme and 4 for a third-order scheme. Larger values of ω bias the MUSCL scheme towards the central difference stencil, and thus are less dissipative. Note that the input value will be limited by DPLR at runtime to lie within the range of possible values in the above equation. A value of approximately 3 is recommended for DPLR at all times; this value will be reset to 2 for second-order simulations.

`(ijk)lim`

This flag specifies the type of flux limiter to employ in the Euler flux extrapolation. Possible choices are:

1	Minmod
2	Superbee
3	Van-Albada

The Minmod limiter (`ijkord` = 1) is recommended for use within DPLR. The others, while somwwhat less dissipative, are also less stable, and should only be employed when low dissipation schemes are actually necessary to obtain highly accurate solutions (such as reactive mixing layer flows).

`(ijk)diss`

This flag specifies the type of eigenvalue limiter to employ in the Euler flux extrapolation. Possible choices are:

0	No added dissipation
1	Standard eigenvalue limiting
2	Standard eigenvalue limiting on linear fields only
3	Standard eigenvalue limiting on non-linear fields only

Eigenvalue limiters are necessary when using Steger-Warming fluxes to prevent glitches at sonic and stagnant points in the flow, and are typically required to obtain robust converged solutions of flows with strong shock waves. Generally speaking eigenvalue limiters should be used in the radial and circumferential flow directions, but should be avoided in the body-normal direction when possible to avoid adding dissipation in the boundary layer. Therefore, the recommended approach is to set `ijkdiss` = 0 in the body-normal direction, and `ijkdiss` = 1 in the other flow directions.

There are some (rare) instances when a normal direction limiter can be very helpful. In this case it is recommended to either try `ijkdiss` = 3, which applies the limiter only to the fluxes with non-linear (u±c) eigenvalues, or to use the `kbl` flag defined previously to turn off application of the normal direction limiter within the boundary layer. However, whenever possible this application should be avoided.

`eps(ijk)`

This flag controls the magnitude of the eigenvalue limiter to employ in the Euler flux extrapolation. In general, values of approximately 0.3 should be employed for hypersonic blunt body flow simulations in the radial and circumferential directions, while 0.0 should be used in the body normal direction. Setting `epsijk` = 0.0 is exactly the same as selecting `ijkdiss` = 0 above. Much lower values of `epsijk` (on the order of 0.01) can be employed in separated flows, which have no strong shocks and are much more sensitive to the effects of added (artificial) dissipation. DPLR will print a run-time warning if it detects a non-zero value of `epsijk` in the body-normal direction in any block.

`iextst`

This flag specifies the time advancement method to employ in the simulation. Possible choices are:

1	Explicit first-order Euler
2	Explicit second-order Runge-Kutta
-1	Implicit data-parallel line relaxation (DPLR)
-2	Implicit data-parallel full matrix (FMDP)

Shockingly enough, the recommended method for steady-state problems is the DPLR method, for which the code is named. For time accurate calculations only the relatively inefficient second-order Runge Kutta (Midpoint) method is offered at this time.

`kmax`

This flag specifies number of implicit relaxation steps to employ when using the DPLR or FMDP methods (`iextst` = -1 or -2). Based on extensive testing during code development, a value of `kmax` = 4 is recommended for all simulations.

`ildir`

This flag specifies direction in which the lines are to be formed for the DPLR method (`iextst` = -1). Possible choices are:

0	Autodetect direction
1	*i*-direction
2	*j*-direction
3	*k*-direction
4	Alternate directions

This flag is only used when `iextst` = -1 (the DPLR method) is selected for time advancement, and controls the direction in which the lines are formed (block tridiagonal solutions is aligned). In general the DPLR method is based on the Gauss-Seidel Line Relaxation (GSLR) method, and the lines should be formed in the body-normal direction for maximum performance. Setting `ildir` = 1, 2, or 3 will cause the code to orient the solver in that block so that lines are formed in the *i, j,* or *k*-directions respectively. If DPLR detects that a line is formed in a non body-normal orientation a warning message will be printed. It is not a fatal error to run DPLR with the lines in non body-normal directions, but the convergence rate and stability of the method will be degraded.

For most problems setting `ildir` = 0 is the best choice. When this option is selected DPLR will automatically determine the best direction to form the lines for each block at runtime by examining the block boundary conditions. For those

blocks for which no body-surface boundary condition is detected the lines will be formed in the *i*-direction. For those blocks with a body surface boundary condition at more than one face the lines will be formed in the direction normal to the last body surface detected.

Finally the user can select `ildir` = 4, which causes DPLR to change the orientation of the lines with each iteration, alternating between *i, j,* and *k*-direction solves. This option was provided mainly for separated flows where there is no preferred direction, but testing to date has not shown there to be a signifcant advantage to using this method.

ibcu

This flag specifies how often to update the implicit boundary conditions during the relaxation process for DPLR or FMDP (`iextst` = -1 or -2). This flag is provided mainly to improve parallel efficiency on machines for which message-passing is very inefficient, and should in general be set to `ibcu` = 1, which forces the implicit boundary conditions to be updated during each relaxation step.

iblag

This flag specifies whether to lag the implicit boundary conditions when using DPLR or FMDP (`iextst` = -1 or -2). Possible choices are:

0	Do not lag implicit BC's
1	Lag implicit BC's

In general it is desired to lag the implicit boundary condition update in order to better mask the message-passing latency and improve parallel performance of the method. However, there are certain instances when the block topology employed makes lagged BC's dangerous. Therefore the default setting within DPLR is `iblag` = 0, which does not mask any of the implicit latency. It is expected that a future upgrade to DPLR will be to automatically determine whether latency can be masked for a given application and the `iblag` flag will be automatically set by the code.

ilt

This flag specifies whether to employ global or local timestepping for implicit simulations. Possible choices are:

-1	Global timestepping
-2	Global timestepping with maximum CFL limit
1	Local timestepping

2 Local timestepping with maximum CFL limit

Testing has shown that local timestepping is extremely unstable when used with DPLR (`iextst` = -1). Therefore only global timestepping should be employed unless `iextst` = -2.

Occasionally one or more blocks of a complex simulation will be much less stable than the rest. One example is the region of the vertical tail in space shuttle simulations. For cases like this the user can specify a maximum CFL number to employ only for the problem blocks by using `ilt` = ±2. The maximum CFL number is set using the `cflm` flag below.

`ibdir`

This flag specifies the grid direction in which to break single block problems for parallel execution. Possible choices are:

1 *i*-direction
2 *j*-direction
3 *k*-direction

Normally the user must use FCONVERT to decompose the problem for parallel execution. The one execption is for simulations in which there is only a single master block with no zonal interfaces (see the FCONVERT users manual for more information). In this case DPLR can perform the necessary decomposition at runtime by breaking the problem into planes in the direction chosen with the `ibdir` flag. Note that DPLR will print a warning message if `ibdir` is set such that the grid is broken in the body-normal direction.

`cflm`

This flag specifies the maximum CFL to use in the current master block. This flag is only used when `ilt` = ±2.

`ibc`

This set of six flags specify boundary condition type to use at each of the six computational cell faces for each grid block. Possible values of `ibc` are:

Basic Boundaries: 0-29
---

0 Pointwise bc read from bc file

1	Fixed at freestream conditions	
2	Fixed at freestream if inflow; extrapolate if outflow	
3	First order extrapolation (supersonic exit)	
4	Second order extrapolation (supersonic exit)	
5	RESERVED	
6	Subsonic reservior inlet; constant mass flow	
7	Periodic	
8	Inviscid wall (flow tangency)	
9	Viscous adiabatic wall	
10	Viscous isothermal wall	
11	180 degree singular axis ($u = -u$)	[3D]
12	180 degree singular axis ($v = -v$)	[3D]
13	180 degree singular axis ($w = -w$)	[3D]
14	Singular x-axis ($v = -v$)	[axi]
15	Singular y-axis ($u = -u$)	[axi]
17	Plane of symmetry ($u = -u$)	
18	Plane of symmetry ($v = -v$)	
19	Plane of symmetry ($w = -w$)	
20	Zone boundary	
21	90 degree singular axis ($v = -v$; $w = -w$)	[3D]
22	90 degree singular axis ($u = -u$; $w = -w$)	[3D]
23	90 degree singular axis ($u = -u$; $v = -v$)	[3D]
24	RESERVED	
25	Catalytic isothermal wall	
26	Catalytic radiative equilibrium wall	
27	Non-catalytic radiative equilibrium wall	

Blowing Wall Boundaries 30-39

30	Viscous isothermal wall with blowing

Slip Wall Boundaries 40-49

40	Viscous isothermal wall with slip
49	Viscous adiabatic wall with slip

Input Profile Boundaries 60-69

---

60 Input primitive variables (ρ_s, *u, v, w, Tv, T*)
61 Input primitive variables (p,c_s[2-ns],u,v,w,Tv,T)
62 Input conserved variables (rho_s,rhou,rhov,rhow,Ev,E)

---
material response coupling bcs 70-79
---

70 -- input species mdots and T; get P from extrapolation, Therm-Eq assumed
71 -- input species cs, mdot, and T; get P from extrapolation, Therm-Eq assumed
75 -- activate surface kinetic mechanism, isothermal (icatmd=1001)
76 -- activate surface kinetic mechanism, rad. eq. (icatmd=1001)

---
subsonic inflow/outflow bcs 80-89
---

81 -- Subsonic reservior inlet; same as #6
82 -- Subsonic inlet; specify mass flow rate & T, extrapolate pressure
85 -- Subsonic exit; specify back (static) pressure, extrapolate others

---
pointwise Twall bcs 100-199
these are always 100 + the corresponding isothermal BC number
---

110 -- no slip isothermal wall (10)
125 -- catalytic isothermal wall (25)
130 -- no slip isothermal wall with blowing (30)
135 -- [NOT WORKING] catalytic isothermal wall with blowing (35)
140 -- isothermal wall with slip (40)
145 -- [NOT WORKING] catalytic isothermal wall with slip (45)

---
pointwise Twall && pointwise blowing bcs 200-299
these are always 200 + the corresponding isothermal BC number
---

230 -- [NOT WORKING] no slip isothermal wall (30)
235 -- [NOT WORKING] catalytic isothermal wall (35)

---

pointwise blowing bcs 300-399
these are always 300 + the corresponding isothermal BC number
---

330 -- [NOT WORKING] no slip isothermal wall (30)
335 -- [NOT WORKING] catalytic isothermal wall (35)

---
bcs 1000-1099
These BC numbers are provided to allow for certain standard BCs
to receive special treatment. Currently supported are 1011:1019 & 1021:1023,
which are standard singular axes or symmetry planes with augmented
eigenvalue limiters in their vicinity, as per the setting of kdg.
When kdg=0 these are indistinguishable from 14:19.

1011 -- 180 degree singular axis (u = -u) [3D]
1012 -- 180 degree singular axis (v = -v) [3D]
1013 -- 180 degree singular axis (w = -w) [3D]
1014 -- singular x-axis (axisymmetric)
1015 -- singular y-axis (axisymmetric)
1017 -- plane of symmetry (u = -u)
1018 -- plane of symmetry (v = -v)
1019 -- plane of symmetry (w = -w)
1021 -- 90 degree singular axis (v = -v; w = -w) [3D]
1022 -- 90 degree singular axis (u = -u; w = -w) [3D]
1023 -- 90 degree singular axis (u = -u; v = -v) [3D]

---
bcs 2000-2099
These BC numbers are provided to allow for certain standard BCs
to receive special treatment. Currently supported are 2011:2015 & 2021:2023,
which are standard singular axes with a maximum CFL limit enforced in their
vicinity (ijk<kdg). The cflm flag is used to set the CFL limit in each block.
When kdg=0 these are indistinguishable from 11:15,21:23.

2011 -- 180 degree singular axis (u = -u) [3D]
2012 -- 180 degree singular axis (v = -v) [3D]
2013 -- 180 degree singular axis (w = -w) [3D]
2014 -- singular x-axis (axisymmetric)
2015 -- singular y-axis (axisymmetric)
2017 -- plane of symmetry (u = -u)
2018 -- plane of symmetry (v = -v)

2019 -- plane of symmetry (w = -w)
2021 -- 90 degree singular axis (v = -v; w = -w) [3D]
2022 -- 90 degree singular axis (u = -u; w = -w) [3D]
2023 -- 90 degree singular axis (u = -u; v = -v) [3D]

Freestream Specification Flags

The following flags define a unique set of flow conditions from which all relevant fluid dynamic quantities can be computed. The user can define any number of freestream blocks (the total number is given by the `nfree` flag above), and can use them to initialize blocks and set freestream boundary conditions on a block-by-block basis, using the `initi` and `ifree` flags, respectively. In all cases input values are in SI units.

`irm`

This flag specifies whether a velocity, Mach number, or unit Reynolds number will be given as input. Possible choices are:

1	Mach number
2	Reynolds number per meter
3	Velocity

The most common (and least ambiguous) input for most free-flight simulations is velocity, because each of the other entries require the velocity to be derived from the thermodynamic and transport models employed in the given simulation. If a Mach number is entered it is assumed to be the equilibrium (as opposed to frozen) value.

`density`

This flag specifies the input freestream mass density.

`M/Re/V`

This flag specifies the input Mach number, unit Reynolds number, or velocity, with the choice determined by the value of irm above. In any case the other two quantities are then determined by DPLR using the input thermodynamic and transport models.

`c(xyz)`

These flags specify the input velocity vector direction cosines. The input values of these flags must be nondimensionalized, i.e. they must satisfy the relation:

$$c_x^2 + c_y^2 + c_z^2 = 1$$

The *u, v*, and *w* components of the velocity vector $\vec{V}$ are then defined simply as:

$$u = c_x \vec{V}; \qquad v = c_y \vec{V}; \qquad w = c_z \vec{V}$$

`Tin, Tvin, Trin, Tein`

These flags specify the input translational, rotational, vibrational, and free electron temperatures, respectively. Note that the number of unique temperatures is dependent on the thermal non-equilibrium models chosen with the `ivib`, `irot`, `ieex`, and `iel` flags above. DPLR will automatically ignore temperatures input for modes that are in equilibrium with another mode of the gas. At the current time free electron non-equilibrium is not supported in DPLR; this flag is included for future expansion and will be silently ignored at runtime.

It is important to note that only one thermal non-equilibrium model may be employed in a given simulation; all blocks are assumed to be governed by the same model. However, each block can have different initial or freestream temperatures simply be defining multiple freestream specifications and using the `initi` and `ifree` flags in each block specification.

`turbi`

This flag is used to specify a freestream turbulence level for the one and two-equation turbulence models. It is not used for laminar or Baldwin-Lomax (algabraic) turbulent simulations. In most cases the default value (0.001) is a good choice; additional information about this parameter is given in the reference manual.

`tkref`

This flag is not currently used in DPLR and is provided for future expansion

`cs`

These flags are an array of input species mass fractions. There must be one entry per species in the chosen chemistry model (as specified in the input `*.chem` file).

All input mass fractions must sum to 1.0 or DPLR will exit with an error message. Note that input of mole fractions is not supported at this time.

CFL Number Listing

The final entries in the DPLR input deck are a list of Courant-Friedreichs-Lewis (CFL) numbers to employed during the simulation. The CFL number is essentially a measure of the explicit inviscid stability limited timestep, and is used by convention in CFD codes to enable time advancement to a steady state solution. For the purposes of DPLR, the CFL number for a given computational cell is defined as the time it takes the fastest wave in the flow to traverse the thinnest dimension of the cell:

$$CFL = \Delta t(u + c)/\Delta\eta$$

The CFL number is turned into a timestep within DPLR:

$$\Delta t = CFL \bullet \Delta\eta/(u + c)$$

but it is important to note that for non-time accurate implicit simulations this is an effective timestep, and may not correlate to actual flow evolution time. This timestep is in general different for every computational cell in the flow. However, for most simulations in DPLR the minum value of Δt at any cell in the flowfield is used for all cells. This is known as global timestepping. It is also possible to use the local value of Δt for each computational cell to advance the solution to steady state; this is known as local timestepping, and is offered as an option in DPLR via the `ilt` flag. However, extensive testing has shown that the data-parallel line relaxation method exhibits much better robustness and convergence rate when global timestepping is employed. Therefore local timestepping is recommended only for simulations usig the full-maxtrix data-parallel method.

Generally speaking larger CFL numbers imply larger timesteps and faster convergence rates. However, for a given problem there is generally a stability limit, or maximum CFL number, that can be used to ensure stable convergence. Employing values above this limit can result in NaN's or solution divergence. Unfortunately there is no ready way to determine the limiting timestep for a given problem *a priori*. This poses somewhat of a conundrum for users, since the convergence rate is very strongly correlated to the CFL number up to this limit. Therefore, for optimum performance it is desireable to run at CFL numbers near, but not over, the stability limit for a given problem. However, with practice users will learn the approximate range of stable CFL numbers for a given class of problem.

Early in the solution the flow is very non-linear, which results in fairly small maximum stable timesteps. However, as the flow evolves towards steady state the maximum timestep typically increases by several orders of magnitude. Therefore, DPLR employs a CFL ramping process, by which the CFL number is increased in a step function every 20 iterations. This is accomplished by typing a list of desired CFL numbers into the input deck. DPLR reads this list at start-up, and then moves to the next number on the list every 20 iterations until it reaches a -1, which signifies

that no more entries are available. At this point DPLR will continue on at the last CFL number until the job is finished. Typically it is preferred to enter CFL number ramps in an approximately exponential fashion, but this is certainly not required.

However, sometimes it is desired to "hold" a constant CFL number for more than 20 timesteps. While in principal this could be accomplished simply by repeating the entry in two or more consecutive lines of the input deck, this could be quite unwieldy in certain situations. Therefore, DPLR offers the alternative that the user can enter a second tab or space separated number on any line with a CFL number. This signifies the number of times to "hold" this value during runtime. For example, the CFL listing:

```
.1
1.
3.      2
10.     2
30.
100.
-1
```

will start at a CFL of 0.1, jump to 1 after 20 timesteps and 3 afer 40 timesteps. It will then hold at three until jumping to a CFL of 10 iteration number 81. This ramping continues until a CFL of 100 is reached.

Each time DPLR is started or restarted the CFL ramp listing is read again from the input deck. Therefore, on a restart, if it is desired to pick up the solution at the final CFL number employed during the previous simulation, the CFL list must be edited. For example, on a restart if the CFL list is edited to look like this:

```
100.
-1
```

DPLR will simply start at a CFL of 100 and stay there for the duration of the simulation. Note that the CFL number can be adjusted while the code is running by using a control (`*.ctrl`) file. See below for more information. Placing a "-1" as the first entry of the listing is a runtime error that will be trapped by DPLR.

It is also possible to enter exact timesteps (Δt) rather than CFL numbers. This is done using negative numbers in the CFL listing. Typically this is only used for time accurate simulations when it is desired to run for a user-specified amount of flow time. CFL numbers and timesteps cannot be mixed and matched in the input listing.

Finally, it is important to note that for most problem the maximum stable timestep is more a function of the physics of the simulation than the computational grid employed. Therefore, the stable CFL number will be a strong function of the wall spacing, since tighter wall spacing will result in larger CFL numbers to achieve the same implicit timestep. For this reason the user may notice a wide range (possibly orders of magnitude) in the maximum stable timstep for a given

vehicle trajectory sweep, while in fact the maximum stable timestep for the set of problems may be more constant.

APPENDIX: Release Notes for Version 3.05.0

UPGRADES:

v3.05.0 -- upgraded/enhanced support of Mitcheltree style catalysis models
-- include electrons in viscosity and thermal conductivity calculations
-- added support of control (*.ctrl) file to change CFL number
-- add uncoupled Spalart-Allmaras turbulence model
-- get 2D SST working (DPLR2D)
-- add automatic grid alignment feature
-- improve NaN trap to work on altix
-- add itshk to cfd input deck
-- add some capability of restarting a turbulent solution with a different turbulence model
-- add Chapman viscosity model
-- write a log file (*.log)
-- added some stuff for MPI-2 compatibility, necessary for proper function on 64-bit systems [provided by Heath Johnson, University of Minnesota]

BUGLIST:

v3.05.0 -- FIXED BUG: small error in Mitcheltree catalysis model
-- FIXED BUG: could not have a catalytic wall unless chemistry was turned on
-- FIXED BUG: not writing correct catalysis constants to restart when material map used
-- FIXED BUG: introduced in v3.04; internal thermal conductivities computed incorrectly for ivib = 4